一、　基本概念

在 Maxima 中，最基本的信息单元是表达式。一个表达式由数、运算符、变量以及常量组成。`2^3^4` 代表 2^{3^4} 即 $2^{(3^4)}$。括号可以改变或者明确运算顺序。

```
(%i1)  2^3^4;
(%o1)
```

$$2417851639229258349412352$$

```
(%i2)  2^(3^4);
(%o2)
```

$$2417851639229258349412352$$

```
(%i3)  (2^3)^4;
(%o3)
```

$$4096$$

输入是一维的，以分号或 `$` 结尾，`$` 表示不返回结果。可以用 `%o3`，`%i4` 之类的符号对以前的输入输出加以引用，`%` 表示上次输出的结果。

用 `Ctrl+D` 快捷键或 `quit()` 语句退出，再次强调，**语句的结尾要加分号或 `$`。**

```
(%i96)  float([%e,%phi,%gamma]);
(%o96)
```

$$[2.7182818, 1.618034, .57721566]$$

常量	符号	说明
%e	e	自然基底
%i	i	虚数单位
%pi	π	圆周率
%phi	ϕ	黄金比
%gamma	γ	欧拉常数
inf	$+\infty$	正无穷
minf	$-\infty$	负无穷

表 1: 常量

1　变量与函数

一个符号被定义，指的是它与一个值或者一段代码建立联系，与一个值建立联系就是变量的赋值，与一段代码建立联系就是函数的定义。赋值操作用“:”连接，函数的定义用“:=”连接。需要注意的是变量的值是多种类型的，不仅仅局限于数，还可以是符号、表、甚至是无名函数，且不需要对变量声明类型。

在定义变量和函数时，重名问题会带来很大麻烦，为了避免重名，常会用到的 `kill(all)$` 语句，其作用就是清除以自定义的变量或函数。

```
(%i97)  if:1;
```

```
Incorrect syntax: : is not a prefix operator
if:1
  ^
(%i98)  If:1;
(%o98)
```

$$1$$

因为 if 是 Maxima 的内置函数，已经有了定义，因此在对其赋值时会发生错误。从上例也可以看出，Maxima 是区分大小写的。

2　原子和表

原子是不可分的单位，包括实数、虚数单位、字串和未定义的符号。包括表、复数在内的任何高级结构最终都是由原子构成的。

表用方括号界定，用逗号分隔各项，是 Maxima 中一种极其重要的数据结构。我们将会逐渐接触到参数表、特征表、函数表、变量表、重表、假设表，甚至表达式和典列也可以看作表。表中套表就构成了重表，矩阵与二重表可以直接相互转化。

Maxima 是用**表处理语言**写的，因此 Maxima 总是喜欢用表结构来存储对象。

```
(%i3)  kill(all)$
(%i1)  [atom(%i),atom(a),(a:sin(x),atom(a)),atom([])];
(%o1)
```

$$[\mathbf{true}, \mathbf{true}, \mathbf{false}, \mathbf{false}]$$

```
(%i2)  [listp(sin(x)),listp([[11,12],[21,22]]),listp([])];
(%o2)
```

$$[\mathbf{false}, \mathbf{true}, \mathbf{true}]$$

我们看到，**方括号语句并行执行，返回所有结果。而圆括号语句顺次执行，只返回最后一个结果。**两种语句的各项用逗号分隔，而且可以嵌套。另外，圆括号语句内可以用 %% 引用上一语句执行的结果。

具有判断对象是否具有某种性质的函数叫做**谓词**。atom 与 listp 都是谓词，大多数谓词的结尾都是 p。

与 Lisp（就是上面提到的表处理语言）不同，空表并非原子。

二、　寻求帮助

```
describe(e);
describe(e,exact);
? e（捷径）
```

直接输出 e 的文档。而

```
describe(e,inexact);
?? e（捷径）
```

则列出包含 e 字眼的所有条目，并标号以待选择。

三、　交互式与批次码

交互式环境与**批次码**环境相对，其特征是**提示符** (%i)。在交互式环境里，一些命令有简化形式，称为**捷径**。在本文中，凡捷径都在其后用括号注明。

一个批次码的例子是前面碰到过的 .dem 文件。另外，圆括号语句、方括号语句也是批次码环境。它们的共同特点是没有提示符，一次执行一批代码。批次码环境不允许走捷径。

除捷径及极少数特殊函数外，绝大多数函数的调用格式都是 **function(args)** 的形式，即使没有参数，括号也不能省略，所以如果看到一个名称后面直接接分号而没有发生错误，说明这个符号不是函数而是变量。

四、　引号操作符

引号操作符有两种，单引号操作符阻止求值，双引号操作符作用相反

```
(%i1)  (a:22,a^2);
(%o1)
```

$$484$$

```
(%i2)  'a^2;
(%o2)
```

$$a^2$$

```
(%i3)  ''%;
(%o3)
```

$$484$$

但是单引号操作符不阻止简化，内置函数对于数的求值也认为是简化。

```
(%i1)  sin(0.1);
(%o1)
```

$$.099833417$$

```
(%i2)  'sin(0.1);
(%o2)
```

$$.099833417$$

```
(%i3)  22^2;
(%o3)
```

$$484$$

```
(%i4)  '22^2;
(%o4)
```

$$484$$

双引号操作符是一种捷径，如果想在批次码环境中使用，应该用其他的形式，如后文中提到的 nouns 开关和 define 函数。

五、　名词与动词

我们把不自动求值的函数或操作符称作**名词**，而把自动求值的函数或操作符称为**动词**。在默认条件下，函数是动词性的。前面提到的单引号的作用就是把动词性的函数或操作符转化为名词，从而达到了阻止函数求值的目的。

nouns 是一个开关，用作环境函数的参数，作用与双引号相同，把名词转化为动词，以使表达式求值。noun 是 declare 函数的一个选项，宣告一个函数为名词。

六、　返回值与附带作用

有的时候我们并不问题特别关心返回值，而更关注于语句的附带作用。比如 kill(all) 的返回值 done 对于我们来说并没有什么意义。我们要的是它的附带作用——清除自定义的变量或函数。因此我们在语句的末尾加 $ 以避免返回结果。所有的语句都有返回值，但并非所有的语句都有附带作用，比如 sqrt(2) 这个语句就没有附带作用。

七、　开关、闸门

Maxima 中的开关（evflag）一般只能取真或假两个值，取真时开关打开，取假时开关关闭。开关用“=”调档，可以把等号想象成一个旋钮，它的档位一般只有两个。个别开关是多值的，称为多掷开关。

闸门（system value）与开关有相似的地方，不过它对以后的所有输入输出都会产生影响。一些闸门是二值的，比如二维闸门（display2d，默认打开），升幂闸门（powerdisp，默认关闭），数值化闸门（numer，默认关闭）等。也有很多闸门是多值的，如域闸门（domain）、带名展开闸门（radexpand），精度闸门（fpprec），显示精度闸门（fpprintprec）。可以认为闸门不仅可以打开关闭还能调节门缝的大小。这些闸门都非常重要，请读者自己试出它们每个的作用。

八、　在特定环境下求值

1　环境函数

环境函数 ev 的作用是：对表达式在诸参数所限定的环境下求值。这些参数可以是开关、修饰函数或者等式。

2　参数是开关

若 x 是开关，则

```
ev(expr,x);
expr,x;（捷径）
ev(expr,x=true);
```

三者等价。也就是说，在环境表达式求值时，开关的默认状态是打开。

闸门 numer 也可以当开关使用

```
(%i5)  properties(numer);
(%o5)
                    [system value, assign property]
```

```
(%i6)  properties(float);
(%o6)
```

$$[\texttt{system value}, \text{transfun}, \text{transfun}, \text{transfun}, \text{evflag}, \text{transfun}, \text{transfun}]$$

```
(%i7)  [ev(exp(3/29),numer),ev(exp(3/29),float)];
(%o7)
```

$$[1.108988430411017, 1.108988430411017]$$

```
(%i8)  [ev(exp(%pi*3/29),numer),ev(exp(%pi*3/29),float)];
(%o8)
```

$$[1.384020049155809, e^{0.103448275862069\,\pi}]$$

```
(%i9)  2*cos(w*t)*3*sin(w*t),exponentialize,expand;
(%o9)
```

$$\frac{3\,i\,e^{-2\,i\,t\,w}}{2} - \frac{3\,i\,e^{2\,i\,t\,w}}{2}$$

3 修饰函数

修饰函数（evfun）可以类比于人的衣服，它不改变对象的本质，只改变对象的形式，后面的“表达式的变换”一节就是修饰函数的大本营。

若一个函数是修饰函数，语句

```
ev(expr,F);
expr,F;
F(ev(expr));
```

等价。修饰函数可以嵌套，例如当 F,G 都是修饰函数时，

```
ev(expr,F,G);
G(F(ev(expr)));
```

等价。

expand,nouns 可以当作修饰函数使用

```
(%i1)  properties(expand);
(%o1)
```

$$[\text{transfun}, \text{transfun}, \text{transfun}, \text{transfun}]$$

```
(%i2)  (a+b)*(c+d),expand;
(%o2)
```

$$b\,d + a\,d + b\,c + a\,c$$

```
(%i3)  ev((a+b)*(c+d),expand);
```

```
(%o3)
```
$$b\,d+a\,d+b\,c+a\,c$$

4 等式

一般情况下，冒号起赋值作用，等号起比较作用。但是在环境函数内部，等号起了代换作用

```
(%i1)  x+y,x=a+y;
(%o1)
```
$$2\,y+a$$

```
(%i2)  %,y=2;
(%o2)
```
$$a+4$$

```
(%i3)  x+y,x=a+y,y=2;
(%o3)
```
$$y+a+2$$

```
(%i4)  x+y,[x=a+y,y=2];
(%o4)
```
$$y+a+2$$

可以看出，代换是并行的。

下面的例子展示了如何验证方程解的正确性

```
(%i5)  eqns:[-2*x-3*y=3,-3*x+2*y=-4]$
(%i6)  solns:solve(eqns);
(%o6)
```
$$\left[\left[y=-\frac{17}{13},x=\frac{6}{13}\right]\right]$$

```
(%i7)  eqns,solns;
(%o7)
```
$$[3=3,-4=-4]$$

下面的例子展示了对 `-1` 的四次方根进行**直角化**和**分式简化**

```
(%i1)  solve(a^4+1);
(%o1)
```
$$\left[a=(-1)^{\frac{1}{4}}\ i,a=-(-1)^{\frac{1}{4}},a=-(-1)^{\frac{1}{4}}\ i,a=(-1)^{\frac{1}{4}}\right]$$

```
(%i2)  %,rectform,ratsimp;
```

```
(%o2)
```
$$\left[a=\frac{\sqrt{2}\,i-\sqrt{2}}{2},a=-\frac{\sqrt{2}\,i+\sqrt{2}}{2},a=-\frac{\sqrt{2}\,i-\sqrt{2}}{2},a=\frac{\sqrt{2}\,i+\sqrt{2}}{2}\right]$$

```
(%i3)  %^4,ratsimp;
(%o3)
```
$$\left[a^4=-1,a^4=-1,a^4=-1,a^4=-1\right]$$

直角化函数与分式简化函数都是修饰函数。

九、　特性宣告

到 atom 和 listp 都是谓词，这类词的结尾大多数是字母 p，下文中出现的 featurep 就是谓词。

```
(%i1)  featurep(expand,evfun);
(%o1)
```
false

```
(%i2)  declare(expand,evfun);
(%o2)
```
done

```
(%i3)  featurep(expand,evfun);
(%o3)
```
false

```
(%i4)  (a+b)^3,expand;
(%o4)
```
$$a^3+3\,a^2\,b+3\,a\,b^2+b^3$$

上面的例子似乎没有意义，因为即使不给展开函数赋予修饰函数的性质，展开函数仍然能够当修饰函数用。但是下面的例子就有了实际意义

```
(%i1)  (declare(aa,bindtest),aa+bb);
  evaluation: unbound variable aa
   -- an error. To debug this try: debugmode(true);
(%i2)  (aa:1,aa+bb);
(%o2)
```
$$\mathrm{bb}+1$$

```
(%i3)  declare(factor,noun)$
(%i4)  factor(x^2+2*x+1);
(%o4)
```
$$\mathrm{factor}\left(x^2+2\,x+1\right)$$

```
(%i5)  %,nouns;
(%o5)
```
$$(x+1)^2$$

```
(%i6)  [H(H(a,b),H(c,H(d,e))),(declare(H,nary),H(H(a,b),H(c,H(d,e))))];
(%o6)
```
$$[H\left(H\left(a,b\right),H\left(c,H\left(d,e\right)\right)\right),H\left(a,b,c,d,e\right)]$$

```
(%i7)  (declare(S,symmetric),[S(b,a),S(a,c,e,d,b)]);
(%o7)
```
$$[S\left(a,b\right),S\left(a,b,c,d,e\right)]$$

```
(%i8)  (declare(T,antisymmetric),[T(b,a),T(a,b,c,e,d),T(a,c,e,d,b)]);
(%o8)
```
$$[-T\left(a,b\right),-T\left(a,b,c,d,e\right),T\left(a,b,c,d,e\right)]$$

赖值（bindtest）性质的变量必须赋值后才能使用。

名词性的函数不会自动求值，必须转化为动词性才会求值。

具有兼爱（nary）特性的函数可以接两个以上的操作数，而这些操作数是可轮换的，比如最常见的乘法运算和加法运算。

对称（symmetric）特性使函数各变量之间具有轮换性质，而反对称（antisymmetric）特性使函数前面的正负号取决于变量是偶排列还是奇排列。

十、　函数表与变量表

1　函数表

functions 是函数表，罗列了当前的**非内置函数**，即用户或附加程序包定义的函数。

```
(%i3)  kill(all)$
(%i1)  functions;
(%o1)
```
$$[]$$

```
(%i2)  (f(x):=sin(x)/x,functions);
(%o2)
```
$$[f\left(x\right)]$$

```
(%i3)  (load(newton),functions);
(%o3)
```
$$[f\left(x\right),\mathrm{newton}\left(\mathrm{expr},\mathrm{guess}\right)]$$

2　变量表

values 是变量表，罗列了所有非内置变量，即由用户或附加程序包定义的变量。

```
(%i17)  kill(all)$
(%i1)  values;
(%o1)
```

$$[]$$

```
(%i2)  ([a:2,b:5,e:x^2/3],values);
(%o2)
```

$$[a, b, e]$$

```
(%i3)  (kill(a),values);
(%o3)
```

$$[b, e]$$

3　移除定义

```
kill(a,b,...);
```

会移除目标 a 和 b 的定义，目标是变量或函数。两个例外是

```
kill(all); 移除所有非内置变量和函数
kill(allbut(a,b,...)); 移除除 a 和 b 之外的所有非内置变量和函数
```

注意移除定义不会对闸门产生影响，亦即，闸门不会恢复到默认状态。

十一、　函数的分配与施用

1　分配

一个符号可以被**分配**（map）到一个表或者表达式

```
(%i4)  kill(all)$
(%i1)  [map('f,[x,y,z]),map('f,x+y+z)];
(%o1)
```

$$[[f(x), f(y), f(z)], f(z) + f(y) + f(x)]$$

```
(%i2)  [map('f,[a*x,b*exp(y),c*log(z)]),map('f,a*x+b*exp(y)+c*log(z))];
(%o2)
```

$$[[f(a\,x), f(b\,e^{y}), f(c\,\log z)], f(c\,\log z) + f(b\,e^{y}) + f(a\,x)]$$

ratsimp 与 map 的连用可以起到对多项式分部简化的效果

```
(%i5)  e:x/(x^2+x)+(y^2+y)/y;
```

```
(%o5)
```
$$\frac{x}{x+x^2}+\frac{y+y^2}{y}$$

```
(%i6)  e,rasimp;
(%o6)
```
$$\frac{x}{x+x^2}+\frac{y+y^2}{y}$$

```
(%i7)  map('ratsimp,e);
(%o7)
```
$$1+\frac{1}{1+x}+y$$

我们再来比较一下 map 与 fullmap 的差别

```
(%i2)  expr:2*%i+3*exp(-4);
(%o2)
```
$$2\,i+3\,e^{-4}$$

```
(%i3)  [map('f,expr),fullmap('f,expr)];
(%o3)
```
$$\left[f\left(2\,i\right)+f\left(3\,e^{-4}\right),f\left(2\right)\,f\left(i\right)+f\left(3\right)\,f\left(e\right)^{f(-4)}\right]$$

2　施用

施用就是把一个列表作为参数交给函数处理。施用的对象只能是一个表，而不能是一个表达式

```
(%i11)  apply('f,[x,y,z]);
(%o11)
```
$$f\left(x,y,z\right)$$

```
(%i12)  apply("+",[x,y,z]);
(%o12)
```
$$x+y+z$$

```
(%i13)  dataL:[[1,2],[2,4]]$
(%i14)  dataM:apply('matrix,dataL);
(%o14)
```
$$\begin{pmatrix}1 & 2\\ 2 & 4\end{pmatrix}$$

```
(%i15)  grind(%)$
  matrix([1,2],[2,4])$
```

扁压函数 grind 实现了一维方式的输出，便于代码的交换。

上面展示了通过**重表**构造矩阵的例子。在 Maxima 中，有很多命令可以对矩阵进行操作。为了节省空间，有时想让矩阵以一维的方式展示出来，这时可以关闭 display2d 闸门。想把矩阵中的重表还原回来，可以使用 args 命令。因此可以认为参数提取是函数施用的逆操作。

```
(%i16)  args(dataM);
(%o16)
```

$$[[1,2],[2,4]]$$

在邮件里附代码的时候，最好使用一维格式。否则，在别人收到的邮件里，上下两行可能会错开或分开，导致难以辨认。

十二、　替换与分拆

1　替换

```
subst(x=a,expr);
subst([x=a,y=b],expr);
```

上式中，被替换的对象 x 和 y 必须是原子或者完整的表达式，如果 x 不满足这个要求，可以使用 ratsubst(x=a,expr) 实现替换。

```
(%i1)  e:f*x^3+g*cos(x);
(%o1)
```

$$g\,\cos x+f\,x^3$$

```
(%i2)  e1:subst(x=a+b,e);
(%o2)
```

$$\cos\left(b+a\right)\,g+\left(b+a\right)^3\,f$$

```
(%i3)  e2:subst(a+b=y,e1);
(%o3)
```

$$g\,\cos y+f\,y^3$$

```
(%i4)  e3:f*x^3+g*cos(y);
(%o4)
```

$$g\,\cos y+f\,x^3$$

```
(%i5)  e4:subst([x=a+b,y=c+d],e3);
(%o5)
```

$$\cos\left(d+c\right)\,g+\left(b+a\right)^3\,f$$

```
(%i6)  subst([a+b=r,c+d=p],e4);
(%o6)
```

$$f\,r^3+g\,\cos p$$

2　分拆和零件替换

一个表达式是一个特殊的表，这个表由两个或两个以上“零件”组成，零件依照**前缀表达式**拼接，并受**升幂闸门**的影响，在升幂闸门打开的条件下，`b+a` 拆为 `[+ a b]`，`a!` 拆为 `[! a]`，`a+c+b` 拆为 `[+ a b c]` 等。因此第一个零件是操作符，后面的零件是操作数。一个大的零件又可分拆为两个或两个以上小的零件，直到每个零件都不可再分为止，比如 `a+2*(b+c)` 最终拆为 `[+ a [* 2 [+ b c]]]`。表中第一个零件编号为零。`part` 的作用是从表达式中分拆出某个零件，而 `substpar` 就是对其中的某个零件进行替换，分拆与替换都要按照层次进行

```
(%i8)  e:a*log(f(y))/(b*exp(f(y)));
(%o8)
```

$$\frac{a\,e^{-f(y)}\,\log f\left(y\right)}{b}$$

```
(%i9)  length(e);
(%o9)
```

$$2$$

```
(%i10)  [part(e,0),part(e,1),part(e,2)];
(%o10)
```

$$\left[/,a\,e^{-f(y)}\,\log f\left(y\right),b\right]$$

```
(%i10)  substpart("+",e,0);
(%o11)
```

$$b+a\,e^{-f(y)}\,\log f\left(y\right)$$

```
(%i12)  length(part(e,1));
(%o12)
```

$$3$$

```
(%i13)  [part(e,1,0),part(e,1,1),part(e,1,2),part(e,1,3)];
(%o13)
```

$$\left[*,a,e^{-f(y)},\log f\left(y\right)\right]$$

```
(%i14)  length(part(e,1,3));
(%o14)
```

$$1$$

```
(%i15)  [part(e,1,3,1,0),part(e,1,3,1,1)];
(%o15)
```

$$[f,y]$$

```
(%i16)  substpart(x,e,1,3,1,1);
(%o16)
```

$$\frac{a\,\log f\left(x\right)\,e^{-f(y)}}{b}$$

正确的使用 `part` 和 `substpart`，我们就可以任意修改一个表达式。

十三、　表达式的变换

1　系数

1.1　提取公因式

```
collectterms(expr,arg1,arg2, ...)
```

例如

```
(%i1)  ex1:a1*(b+c/2)^2+a2*(d+e/3)^3,expand;
(%o1)
```

$$\frac{\mathrm{a2}\,e^3}{27}+\frac{\mathrm{a2}\,d\,e^2}{3}+\mathrm{a2}\,d^2\,e+\mathrm{a2}\,d^3+\frac{\mathrm{a1}\,c^2}{4}+\mathrm{a1}\,b\,c+\mathrm{a1}\,b^2$$

我们怎么把展开后的式子恢复到展开前呢？我们先提取公因式，然后再用因式分解并分配到各项

```
(%i2)  collectterms(ex1,a1,a2);
(%o2)
```

$$\mathrm{a2}\left(\frac{e^3}{27}+\frac{d\,e^2}{3}+d^2\,e+d^3\right)+\mathrm{a1}\left(\frac{c^2}{4}+b\,c+b^2\right)$$

```
(%i3)  map('factor,%);
(%o3)
```

$$\frac{\mathrm{a2}\,(e+3\,d)^3}{27}+\frac{\mathrm{a1}\,(c+2\,b)^2}{4}$$

由于 Maxima 的简化规则阻止我们进一步把 $3^3=27$ 和 $2^2=4$ 弄到括号里面去，所以我们也就不费劲地做多次替换以得到最原始的式子了。

1.2　直接提取某项的系数

```
coeff(expr,x,n);
```

返回式 expr 中 x^n 项的系数。如果省略 n，将返回 x 项的系数，即第三个参数的缺省值为 1。x 是表达式中的一个原子或者一个完整的表达式，例如 sin(y)、a[j] 或者 (a+b)。有的时候需要对表达式进行展开或分解以使 x^n 项明朗化，而这些，coeff 并不会自动为我们做。

```
(%i1)  expr:a^2+2*a*b+b^2;
(%o1)
```

$$b^2+2\,a\,b+a^2$$

```
(%i2)  coeff(expr,a+b,2);
(%o2)
```

$$0$$

```
(%i3)  (factor(expr),coeff(%%,a+b,2));
(%o3)
```

$$1$$

1.3 变换后提取系数

从上面的例子可以看出，coeff 是很笨的，它不对表达式进行变换，而是直接提取系数，而 ratcoef[1]则要聪明很多，它先对表达式作变换（分解或展开），然后再提取 x^n 项的系数

```
(%i1)  (ex2:(a*x+b)^2,[coeff(ex2,x),ratcoef(ex2,x),ratcoef(ex2,x,0)]);
(%o1)
```

$$[0, 2\,a\,b, b^2]$$

```
(%i2)  ex3:x^2+2*x*y+y^2$
(%i3)  [coeff(ex3,x+y,2),ratcoef(ex3,x+y,2),ratcoef(ex3,(x+y)^2),ratcoef(ex3,2*x)];
(%o3)
```

$$[0, 1, 1, y]$$

可以对方程提取系数

```
(%i19)  eqn:(a*sin(x)+b*cos(x))^3=c*sin(x)*cos(x);
(%o19)
```

$$(b\,\cos x + a\,\sin x)^3 = c\,\cos x\,\sin x$$

```
(%i19)  ratcoef(eqn,sin(x),0);
(%o20)
```

$$b^3\,(\cos x)^3 = 0$$

```
(%i21)  ratcoef(eqn,sin(x));
(%o21)
```

$$3\,a\,b^2\,(\cos x)^2 = c\,\cos x$$

2 分式的展通裂解简

分式指**多项式之比及这种比值的和。**

我们可以用 diff(expr,x) 求一阶导数，用 diff(expr,x,2) 来求的二阶导数等等

```
(%i22)  de1:diff((x+3)^10,x);
(%o22)
```

$$10\,(3+x)^9$$

由于上面的表达式是一个关于 x 的多项式，所以我们也能用 ratdiff(expr,x) 来实现

```
(%i23)  de1r:ratdiff((x+3)^10,x);
(%o23)
```

$$196830 + 590490\,x + 787320\,x^2 + 612360\,x^3 + 306180\,x^4 + 102060\,x^5 + 22680\,x^6 + 3240\,x^7 + 270\,x^8 + 10\,x^9$$

```
(%i24)  factor(de1r);
(%o24)
```

$$10\,(3+x)^9$$

[1]也作 ratcoeff

下面我们展示三种展开多项式的工具

```
(%i25)  e:(x+3)^10;
(%o25)
```

$$(3+x)^{10}$$

```
(%i26)  rat(e);
(%o26)
```

$$59049+196830\,x+295245\,x^2+262440\,x^3+153090\,x^4+61236\,x^5+17010\,x^6+3240\,x^7+405\,x^8+30\,x^9+x^{10}$$

```
(%i27)  ratexpand(e);
(%o27)
```

$$59049+196830\,x+295245\,x^2+262440\,x^3+153090\,x^4+61236\,x^5+17010\,x^6+3240\,x^7+405\,x^8+30\,x^9+x^{10}$$

```
(%i28)  expand(e);
(%o28)
```

$$59049+196830\,x+295245\,x^2+262440\,x^3+153090\,x^4+61236\,x^5+17010\,x^6+3240\,x^7+405\,x^8+30\,x^9+x^{10}$$

```
(%i29)  factor(%);
(%o29)
```

$$(3+x)^{10}$$

如果想让多项式升幂排列，可以把闸门 powerdisp 打开

```
(%i1)  [x+y,a1+a2,1+x+x^2];
(%o1)
```

$$[y+x,\mathrm{a2}+\mathrm{a1},x^2+x+1]$$

```
(%i2)  (powerdisp:true,[x+y,a1+a2,1+x+x^2]);
(%o2)
```

$$[x+y,\mathrm{a1}+\mathrm{a2},1+x+x^2]$$

下面我们研究一下分式的和

```
(%i1)  expr:(x-1)/(x+1)^2+1/(x-1);
(%o1)
```

$$\frac{x-1}{(x+1)^2}+\frac{1}{x-1}$$

```
(%i2)  expand(expr);
(%o2)
```

$$\frac{x}{x^2+2\,x+1}-\frac{1}{x^2+2\,x+1}+\frac{1}{x-1}$$

```
(%i3)  ratexpand(expr);
```

(%o3)

$$\frac{2\,x^2}{x^3+x^2-x-1}+\frac{2}{x^3+x^2-x-1}$$

我们看到 ratexpand 把分式通分了，expand 却没有。这证明，分式类函数在处理分式方面更加高效。事实确实如此，对于一个非常庞大的分式，分式类函数往往能够瞬间得到结果，用 expand 处理却几分钟仍不见结果。

expand 的一般形式 expand(expr,p,n) 可以控制展开分式的哪一部分。对应地，rat 的一般形式是 rat(expr,x_1, ..., x_n)，其作用也可类比得到。

2.1 裂项

前面我们已经领教了 ratsimp 的通分简化功能，可以认为与其具有相反用途的是裂项函数 partfrac。

```
partfrac(expr,var);
```

```
(%i12)  e:1/(1+x)^2-2/(1+x)+2/(2+x);
```
(%o12)

$$\frac{2}{x+2}-\frac{2}{x+1}+\frac{1}{(x+1)^2}$$

```
(%i13)  e,ratsimp;
```
(%o13)

$$-\frac{x}{x^3+4\,x^2+5\,x+2}$$

```
(%i14)  partfrac(%,x);
```
(%o14)

$$\frac{2}{x+2}-\frac{2}{x+1}+\frac{1}{(x+1)^2}$$

3 带名表达式的简化、带名类函数

前面我们遇到的表达式大多都是分式，分式类函数可以很好的处理这类式子。但是对于对数、指数、根式等，分式类函数就显得力不从心了。我们知道，表达式简化的焦点在于括号的展开与合并，而这些表达式的一个共同特点，就是括号的外面多了一个函数名，正是这个本质的区别导致了分式类函数的失效。根据这个观点，我们提炼出**带名**的概念，下文出现的 rad 前缀，就可以认为是带名之意。而**带名类函数**指的是专门处理带名表达式的函数。

```
(%i13)  expr:(exp(x)-1)/(exp(x/2)+1);
```
(%o13)

$$\frac{e^x-1}{e^{\frac{x}{2}}+1}$$

```
(%i14)  expr,ratsimp;
```
(%o14)

$$\frac{e^x-1}{e^{\frac{x}{2}}+1}$$

```
(%i15)  expr,radcan;
(%o15)
```

$$e^{\frac{x}{2}}-1$$

```
(%i16)  logexpr:log((x+1)*(x-2))+log(x);
(%o16)
```

$$\log\left((x-2)\,(x+1)\right)+\log x$$

```
(%i17)  logexpr,ratsimp;
(%o17)
```

$$\log\left(x^2-x-2\right)+\log x$$

```
(%i18)  logexpr,fullratsimp;
(%o18)
```

$$\log\left(x^2-x-2\right)+\log x$$

```
(%i19)  logexpr,radcan;
(%o19)
```

$$\log\left(x+1\right)+\log x+\log\left(x-2\right)$$

```
(%i20)  %,logcontract;
(%o20)
```

$$\log\left(x^3-x^2-2\,x\right)$$

```
(%i21)  sqrt(2)*sqrt(3),radcan;
(%o21)
```

$$\sqrt{2}\,\sqrt{3}$$

```
(%i22)  sqrt(2)*sqrt(3),rootscontract;
(%o22)
```

$$\sqrt{6}$$

```
(%i23)  sqrt(6)*sqrt(3),radcan;
(%o23)
```

$$3\,\sqrt{2}$$

```
(%i24)  sqrt(6)*sqrt(3),rootscontract;
(%o24)
```

$$3\,\sqrt{2}$$

```
(%i25)  sqrt(6)/sqrt(3),radcan;
(%o25)
```

$$\sqrt{2}$$

```
(%i26)  sqrt(6)/sqrt(3),rootscontract;
(%o26)
```

$$\sqrt{2}$$

4　三角函数的简化

三角函数式不能用带名类函数处理。专门处理三角函数式的相关函数有

```
trigsimp,tirgexpand,trigreduce,trigrat
```

trigsimp 将其他三角函数转化为正弦或余弦，并且会利用到 $\sin^2 x + \cos^2 x = 1$ 这一规则

```
(%i1)  trigsimp(tan(x));
(%o1)
```

$$\frac{\sin x}{\cos x}$$

```
(%i2)  sin(x+y),trigexpand;
(%o2)
```

$$\cos x\ \sin y + \sin x\ \cos y$$

```
(%i3)  x+3*cos(x)^2-sin(x)^2,trigreduce;
(%o3)
```

$$\frac{\cos\left(2\,x\right)}{2} + 3\left(\frac{\cos\left(2\,x\right)}{2} + \frac{1}{2}\right) + x - \frac{1}{2}$$

```
(%i4)  trigrat(sin(3*a)/sin(a+%pi/3));
(%o4)
```

$$\sqrt{3}\,\sin\left(2\,a\right) + \cos\left(2\,a\right) - 1$$

trigsimp，trigreduce，ratsimp 和 radcan 几个函数连用有可能进一步简化结果。

```
(%i52)  rtt(e):=radcan(trigrat(trigsimp(e)))$
```

如果式中只有正弦余弦函数，这个函数与 trigreduce 是等价的，否则两者会返回不同的结果。

trigreduce 与 trigexpand 两个函数的作用是相反的

```
(%i1)  e:sin(x+y),trigexpand;
(%o1)
```

$$\cos x\ \sin y + \sin x\ \cos y$$

```
(%i2)  e,trigreduce;
(%o2)
```

$$\sin\left(y + x\right)$$

```
(%i3)  rtt(e);
(%o3)
```

$$\mathrm{rtt}\left(\cos x\ \sin y + \sin x\ \cos y\right)$$

```
(%i4)  e:tan(x+y),trigexpand;
(%o4)
```

$$\frac{\tan y+\tan x}{1-\tan x\,\tan y}$$

```
(%i5)  e,trigreduce;
(%o5)
```

$$-\frac{\tan y}{\tan x\,\tan y-1}-\frac{\tan x}{\tan x\,\tan y-1}$$

```
(%i6)  %,ratsimp;
(%o6)
```

$$-\frac{\tan y+\tan x}{\tan x\,\tan y-1}$$

```
(%i7)  rtt(e);
(%o7)
```

$$\mathrm{rtt}\left(\frac{\tan y+\tan x}{1-\tan x\,\tan y}\right)$$

```
(%i8)  e:cosh(x+y),trigexpand;
(%o8)
```

$$\sinh x\,\sinh y+\cosh x\,\cosh y$$

```
(%i9)  e,trigreduce;
(%o9)
```

$$\cosh\left(y+x\right)$$

```
(%i10)  rtt(e);
(%o10)
```

$$\mathrm{rtt}\left(\sinh x\,\sinh y+\cosh x\,\cosh y\right)$$

```
(%i11)  expand(%);
(%o11)
```

$$\mathrm{rtt}\left(\sinh x\,\sinh y+\cosh x\,\cosh y\right)$$

在下面的例子中，注意 `trigexpand(expr)` 和 `expr,trigexpand` 的不同。这种不同的根源是 `tirgexpand` 既是修饰函数，又是开关

```
(%i1)  e1:trigexpand(tan(2*x+y));
(%o1)
```

$$\frac{\tan y+\tan\left(2\,x\right)}{1-\tan\left(2\,x\right)\,\tan y}$$

```
(%i2)  e2:tan(2*x+y),trigexpand;
(%o2)
```

$$\frac{\tan y+\frac{2\,\tan x}{1-\left(\tan x\right)^2}}{1-\frac{2\,\tan x\,\tan y}{1-\left(\tan x\right)^2}}$$

```
(%i3)  rtt(e1);
(%o3)
```

$$\mathrm{rtt}\left(\frac{\tan y+\tan\left(2\,x\right)}{1-\tan\left(2\,x\right)\,\tan y}\right)$$

```
(%i4)  rtt(e2);
(%o4)
```

$$\mathrm{rtt}\left(\frac{\tan y+\frac{2\,\tan x}{1-\left(\tan x\right)^2}}{1-\frac{2\,\tan x\,\tan y}{1-\left(\tan x\right)^2}}\right)$$

```
(%i5)  trigsimp(e1);
(%o5)
```

$$-\frac{\cos\left(2\,x\right)\,\sin y+\sin\left(2\,x\right)\,\cos y}{\sin\left(2\,x\right)\,\sin y-\cos\left(2\,x\right)\,\cos y}$$

```
(%i6)  trigsimp(e2);
(%o6)
```

$$-\frac{\left(2\,\left(\cos x\right)^2-1\right)\,\sin y+2\,\cos x\,\sin x\,\cos y}{2\,\cos x\,\sin x\,\sin y+\left(1-2\,\left(\cos x\right)^2\right)\,\cos y}$$

5　复数

操作复数的相关函数有

```
realpart,imagpart,rectform,polarform,abs,carg
```

```
(%i1)  cform:[cos(x),sin(x),cosh(x)],exponentialize;
(%o1)
```

$$\left[\frac{e^{i\,x}+e^{-i\,x}}{2},-\frac{i\,\left(e^{i\,x}-e^{-i\,x}\right)}{2},\frac{e^x+e^{-x}}{2}\right]$$

```
(%i2)  cform,demoivre;
(%o2)
```

$$\left[\cos x,\sin x,\frac{e^x+e^{-x}}{2}\right]$$

```
(%i3)  cform,rectform;
(%o3)
```

$$\left[\cos x,\sin x,\frac{e^x+e^{-x}}{2}\right]$$

```
(%i4)  realpart(cform);
(%o4)
```

$$\left[\cos x,\sin x,\frac{e^x+e^{-x}}{2}\right]$$

```
(%i5)  imagpart(cform);
(%o5)
```

$$[0,0,0]$$

6　判零函数

判零函数 `zeroequiv(expr,var)` 用若干值域内的点来验证表达式 `expr` 是否恒等于零。例如验证下式是否成立

$$\cos^2(x-1)=\frac{1}{2}\sin(2)\sin(2\,x)+\cos(2)\cos(2\,x)+1$$

虽然用其他工具也可以做到这一点，但是这里我们不妨用判零函数验证。

```
(%i1)  e1:cos(x-1)^2;
(%o1)
```

$$(\cos(x-1))^2$$

```
(%i2)  e2:(sin(2)*sin(2*x)+cos(2)*cos(2*x)+1)/2;
(%o2)
```

$$\frac{\sin 2\,\sin(2\,x)+\cos 2\,\cos(2\,x)+1}{2}$$

```
(%i3)  zeroequiv(e1-e2,x);
(%o3)
```

true

十四、　积分

1　得到解析结果

```
(%i1)  e:x/(x^3+1);
(%o1)
```

$$\frac{x}{x^3+1}$$

```
(%i2)  ie:integrate(e,x);
(%o2)
```

$$\frac{\log(x^2-x+1)}{6}+\frac{\arctan\left(\frac{2\,x-1}{\sqrt{3}}\right)}{\sqrt{3}}-\frac{\log(x+1)}{3}$$

```
(%i3)  diff(ie,x);
(%o3)
```

$$\frac{2}{3\left(\frac{(2\,x-1)^2}{3}+1\right)}+\frac{2\,x-1}{6\,(x^2-x+1)}-\frac{1}{3\,(x+1)}$$

```
(%i4)  %,ratsimp;
(%o4)
```

$$\frac{x}{x^3+1}$$

微分是积分的逆运算，因此，积分结果的正确性往往可以用微分来检验。

再来看一下什么情况下可以用 ratdiff 来检验

```
(%i1)  ratdiff(sin(x),x);
   `ratdiff' variable is embedded in kernel
    -- an error. To debug this try: debugmode(true);
(%i2)  ratdiff(sin(x)^2,sin(x));
(%o2)
```

$$2\,\sin x$$

在做积分的时候，有时会询问某个参数的取值，这是必要的，因为不同的取值往往会影响积分结果。由于 Maxima 在积分前会首先查询**假设表**，因此我们可以通过事先添加假设的方法避免积分时的问询。添加假设使用 assume 函数，删除假设使用 forget 函数，查看假设表则使用 facts() 函数。

```
(%i1)  facts();
(%o1)
```

$$[]$$

```
(%i1)  integrate(x*exp(-a*x)*cos(w*x),x,0,inf);
   Is  a  positive, negative, or zero?
   p;
(%o2)
```

$$-\frac{w^2-a^2}{w^4+2\,a^2\,w^2+a^4}$$

```
(%i3)  (assume(a>0),facts());
(%o3)
```

$$[a>0]$$

```
(%i4)  integrate(x*exp(-a*x)*cos(w*x),x,0,inf);
(%o4)
```

$$-\frac{w^2-a^2}{w^4+2\,a^2\,w^2+a^4}$$

```
(%i5)  (forget(a>0),facts());
(%o5)
```

$$[]$$

2 得到数值结果

闭区间上的定积分有两种选择：一种是先求出原函数，再代入数值，最后将结果数值化，这种方法用到 integrate 和 float 函数；另一种是直接使用数值方法积分，用到 quad_qags 函数。为更好地比较这两种方法，我们做一个难一点的定积分，积分的结果含有一个**特殊函数**

```
(%i1)  is:integrate(exp(x^3),x,1,2);
(%o1)
```

$$\frac{\left(\sqrt{3}\,i-1\right)\,\left(\text{gamma\_incomplete}\left(\frac{1}{3},-8\right)-\text{gamma\_incomplete}\left(\frac{1}{3},-1\right)\right)}{6}$$

```
(%i2)  float(is),fpprintprec:8;
(%o2)
```

$$.16666667\,\left(-719.2849\,i-1.0\,\left(.66608191-3.4863699\,i\right)-412.60039\right)\,\left(1.7320508\,i-1.0\right)$$

```
(%i3)  ival:expand(%);
(%o3)
```

$$275.51098$$

由于第一种方法用到了数值化函数 float，因此我们把这种方法称作数值化方法。我们以 bfloat,fpprec=20 的结果作为“准确值”，看一下数值化方法的误差

```
(%i4)  expand(bfloat(is)),fpprec:20;
(%o4)
```

$$5.7824115865893569814_B\times 10^{-19}\,i+2.7551098376331160126_B\times 10^{2}$$

```
(%i5)  tval20:realpart(%);
(%o5)
```

$$2.7551098376331160126_B\times 10^{2}$$

```
(%i6)  abs(ival-tval20),fpprec:20;
(%o6)
```

$$5.4085366393841205479_B\times 10^{-11}$$

可以看到，数值化方法得到的结果有十二位精度。

我们再用数值积分方法（quad_qags）计算一次，这个函数的返回格式是 **[积分结果，估计误差，取样数目，出错代码]**。

```
(%i7)  quad_qags(exp(x^3),x,1,2);
(%o7)
```

$$\left[275.51098,3.23056159\times 10^{-7},21,0\right]$$

```
(%i8)  abs(first(%)-tval20),fpprec=20;
(%o8)
```

$$2.7275057212783337945_B\times 10^{-14}$$

这种方法的精度达到了十五位。

在内存允许的情况下，通过 bfloat 函数，Maxima 能够进行**任意精度**的计算。

不要害怕，Maxima 能够瞬间计算出 2^{20000}，甚至计算 $2^{2000000}$ 也只需片刻等待。

```
(%i3)  2^20000;
(%o3)
```

32733906078961418700131896968275991522166420460430647894832913680961337964046745548832700923259041

```
(%i4)  [bfloat(1/2^20000),float(1/2^20000)];
(%o4)
```

$$\left[3.054936363499605_B \times 10^{-151}, 3.0549363634996 \times 10^{-151}\right]$$

十五、　解方程

1　求解析解

```
solve(expr,x);
solve(expr);
```

式中 expr 是一个方程，否则，假定为 expr=0。x 是变量或函数，两个参数都可以是表，返回结果是根的列表。

```
(%i1)  solve(x^2+1);
(%o1)
```

$$[x = -i, x = i]$$

```
(%i2)  solve(x+a);
  More unknowns than equations - `solve'
  Unknowns given :
  ** error while printing error message **
  More unknowns than equations - `solve'~
                  ~%Unknowns given :  ~%~M~
                  ~%Equations given:  ~%~M
   -- an error. To debug this try: debugmode(true);
(%i3)  solve(x+a,x);
(%o3)
```

$$[x = -a]$$

```
(%i4)  solve(sin(x)^2+sin(x)+1,sin(x));
(%o4)
```

$$\left[\sin x = -\frac{\sqrt{3}\,i+1}{2}, \sin x = \frac{\sqrt{3}\,i-1}{2}\right]$$

```
(%i5)  solve((x+1)^2+(x+1)+1,x+1);
```

```
(%o5)
```
$$[]$$

```
(%i6)  solve([x+y=2,x-y=1],[x,y]);
(%o6)
```
$$\left[\left[x=\frac{3}{2},y=\frac{1}{2}\right]\right]$$

```
(%i7)  solve(sin(x)+0.5);
  rat: replaced 0.5 by 1/2 = 0.5
  solve: using arc-trig functions to get a solution.
  Some solutions will be lost.
(%o7)
```
$$\left[x=-\frac{\pi}{6}\right]$$

```
(%i8)  solve(sin(x)+cos(x)=sqrt(2));
(%o8)
```
$$\left[\sin x=\sqrt{2}-\cos x\right]$$

```
(%i9)  solve(x^5+3*x^3-2*x^2+x-1);
(%o9)
```
$$\left[0=x^5+3\,x^3-2\,x^2+x-1\right]$$

```
(%i10)  eqn:x^6-1=0$
(%i11)  solns:solve(eqn);
(%o11)
```
$$\left[x=\frac{\sqrt{3}\,i+1}{2},x=\frac{\sqrt{3}\,i-1}{2},x=-1,x=-\frac{\sqrt{3}\,i+1}{2},x=-\frac{\sqrt{3}\,i-1}{2},x=1\right]$$

```
(%i12)  for i thru length(solns) do
  disp(ev(eqn,solns[i],ratsimp))$
  0=0
  0=0
  0=0
  0=0
  0=0
  0=0
```

可以看到，`solve` 找到的是解析解，解析解的好处是没有误差。但 `solve` 不是智能的，它依据的是人给它制定好的规则，比如二、三、四次方程的求根公式、特殊角的三角函数对应表等。所以不要指望依靠 `solve` 找到一般的**高次方程**[2]及**超越方程**的解析解，它甚至连 $\sin(x)+\cos(x)=\sqrt{2}$ 这样的方程都解不出。由于解析解不总是存在，且我们不总需要解析解，这时我们可以用 `allroots` 找到方程的数值解。当然，对于包含异名三角函数的可以求得解析解的方程，我们还是有办法的

[2]指五次及五次以上的代数方程，这类方程没有求根公式

```
(%i1)  fpprintprec:8$
(%i2)  allroots(x^5+3*x^3-2*x^2+x-1);
(%o2)
```

$$[x = .67804805\,i - .040188631, x = -.67804805\,i - .040188631, x = .72086168, x = 1.7041836\,i - .32024221, x =$$

```
(%i3)  allroots(x^6-1);
(%o3)
```

$$[x = 0.8660254\,i + 0.5, x = 0.5 - 0.8660254\,i, x = 0.8660254\,i - 0.5, x = -0.8660254\,i - 0.5, x = 1.0, x = -1.0]$$

数值方法可以解更多的方程，但是结果一般存在误差，误差大小取决于计算精度。

2 求实根

对于方程 $275e^{-r} + 275e^{-2r} + 275e^{-3r} + 275e^{-4r} + 5275e^{-5r} = 4750$，我们知道，单纯使用 `solve` 是无法求解的。这里介绍一种方法：用 `z` 代换方程中的 `exp(r)`，但是代换之后得到的五次方程仍然无法用 `solve` 解出。假如我们只关心方程的实数解，我们可以用 `realroots` 求解

```
(%i1)  eqn:275*exp(-r)+275*exp(-2*r)+275*exp(-3*r)+275*exp(-4*r)+5275*exp(-5*r)=4750;
(%o1)
```

$$275\,e^{-r} + 275\,e^{-2\,r} + 275\,e^{-3\,r} + 275\,e^{-4\,r} + 5275\,e^{-5\,r} = 4750$$

```
(%i2)  zeqn:ratsubst(z,exp(r),eqn);
(%o2)
```

$$\frac{275\,z^4 + 275\,z^3 + 275\,z^2 + 275\,z + 5275}{z^5} = 4750$$

```
(%i3)  soln:solve(zeqn);
(%o3)
```

$$\left[0 = 190\,z^5 - 11\,z^4 - 11\,z^3 - 11\,z^2 - 11\,z - 211\right]$$

```
(%i4)  rr:realroots(soln[1]);
(%o4)
```

$$\left[z = \frac{35805935}{33554432}\right]$$

```
(%i5)  zeqn,rr,ratsimp;
(%o5)
```

$$\frac{1118223606694409919034751162905076642611 2}{2354155174380926220896751357249338375} = 4750$$

```
(%i6)  %,float;
(%o6)
```

$$4749.9996 = 4750$$

这个解的精度太低了，为了提高精度，我们修改 `rootsepsilon` 的值

```
(%i7)  rr:realroots(soln[1],1.0e-16);
```

```
(%o7)
```

$$\left[z = \frac{38446329182573765}{36028797018963968}\right]$$

```
(%i8)  zeqn,rr,numer;
(%o8)
```

$$4750.0 = 4750$$

由于 $z = e^r$，所以 $r = \ln z$

```
(%i9)  rval:log(rhs(rr[1])),numer;
(%o9)
```

$$.064944686$$

```
(%i10)  eqn,r=rval;
(%o10)
```

$$4750.0 = 4750$$

上例中 rhs 意为取方程右侧 (right-hand-side)。

3 区间内寻根

find_root(expr,var,a,b)

返回方程 expr=0 在 a 到 b 之间的数值解。其中，expr 是变量 var 的函数。

我们可以用 find_root 解一些 solve 所不能解的超越方程。a 与 b 值的确定可以通过函数图像估出

```
(%i11)  e:sin(x)-x/2;
(%o11)
```

$$\sin x - \frac{x}{2}$$

```
(%i12)  solve(e);
(%o12)
```

$$[x = 2\ \sin x]$$

```
(%i13)  plot2d(e,[x,0,2*%pi],[style,[lines,5]],[ylabel," "],[xlabel,"sin(x)-x/2"],
  [gnuplot_preamble,"set zeroaxis lw 2"])$
```

Maxima 调用 Gnuplot 画图，最后面的参数就是传递给 Gnuplot 的。从图中我们可以看出有一个根在 1 到 3 之间

```
(%i14)  xr:find_root(e,x,1,3);
(%o14)
```

$$1.8954943$$

```
(%i15)  e,x=xr;
(%o15)
```

$$0.0$$

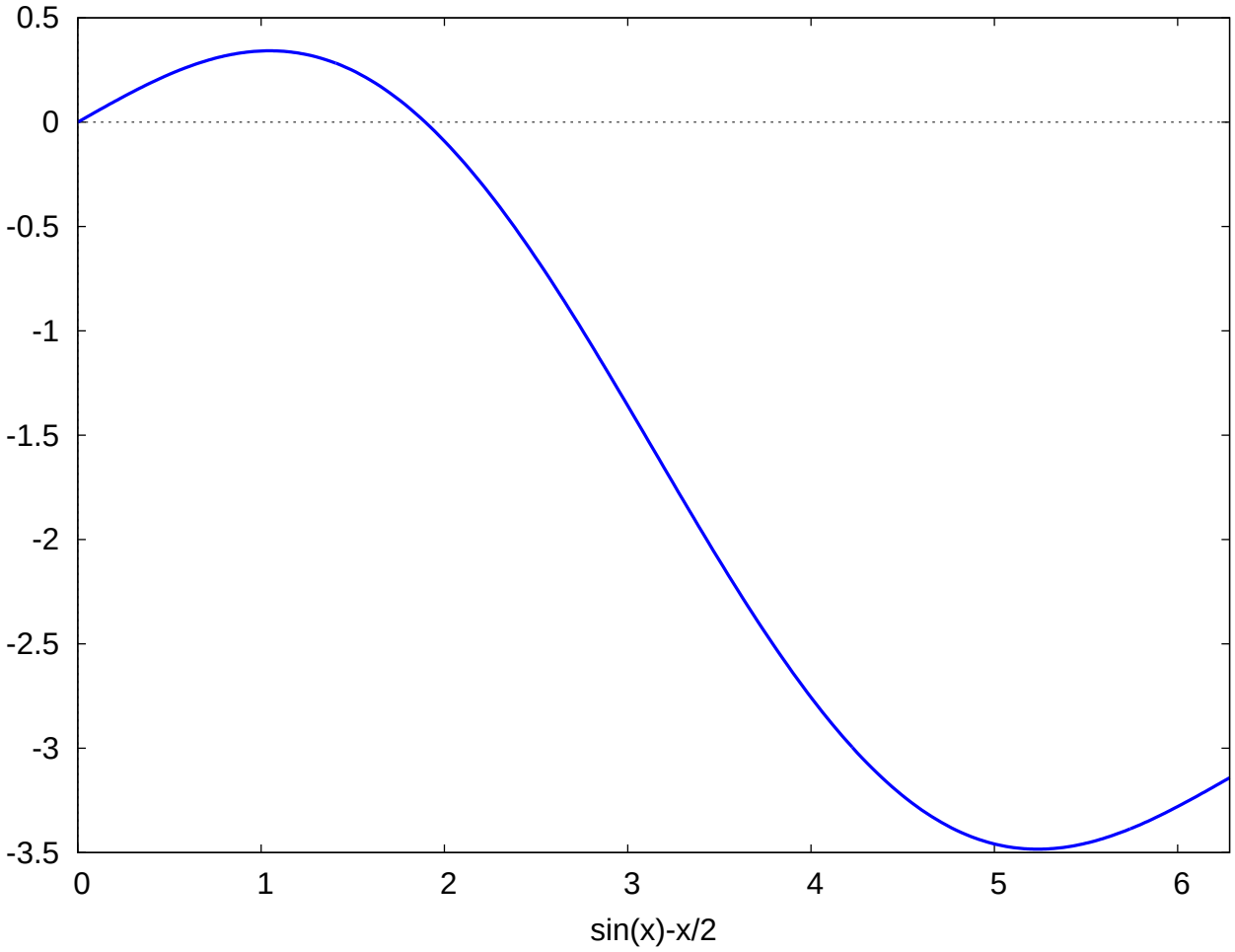

图 1: $\sin(x) - x/2$

4 间接求得解析解

包含异名三角函数的方程如

$$a(1 - \sin x) = 2b\cos x$$

一般情况下，`solve` 对这类方程是无能为力的，但是有多种方法可以帮助 `solve` 得到我们想要的解。

4.1 指数化

```
(%i1)  eqn:a*(1-sin(x))-2*b*cos(x)=0$
(%i2)  eqn,exponentialize;
(%o2)
```

$$a\left(\frac{i\left(e^{ix}-e^{-ix}\right)}{2}+1\right)-b\left(e^{ix}+e^{-ix}\right)=0$$

```
(%i3)  solns:solve(%,x);
(%o3)
```

$$\left[x=-i\log\left(\frac{2\,i\,b}{2\,b-i\,a}+\frac{a}{2\,b-i\,a}\right),x=-i\log\left(\frac{a}{2\,b-i\,a}-\frac{2\,i\,b}{2\,b-i\,a}\right)\right]$$

```
(%i4)  solns:solns,rectform,ratsimp;
```
(%o4)
$$\left[x=\frac{\pi}{2},x=-\operatorname{atan2}\left(\frac{4\,b^2-a^2}{4\,b^2+a^2},\frac{4\,a\,b}{4\,b^2+a^2}\right)\right]$$

```
(%i5)  eqn,solns[1],ratsimp;
```
(%o5)
$$0=0$$

```
(%i6)  eqn,solns[2],ratsimp;
```
(%o6)
$$0=0$$

```
(%i7)  x2:atan2(a^2-4*b^2,4*a*b)$
(%i8)  eqn,x=x2,ratsimp;
```
(%o8)
$$-\frac{a\,\sqrt{16\,b^4+8\,a^2\,b^2+a^4}-4\,a\,b^2-a^3}{4\,b^2+a^2}=0$$

```
(%i9)  scanmap('factor,%);
```
(%o9)
$$0=0$$

例中的 `atan2(x,y)` 表示 $\arctan\dfrac{x}{y}$

4.2　联立方程组

`solve` 不会利用 $\cos^2 x+\sin^2 x=1$ 这个关系，我们可以把这个方程与原方程联立，再用 `solve` 解联立的方程组。

```
(%i10)  eqns:[a*(1-sin(x))-2*b*cos(x)=0,cos(x)^2+sin(x)^2=1]$
(%i11)  solns:solve(eqns,[sin(x),cos(x)]);
```
(%o11)
$$\left[[\sin x=1,\cos x=0],\left[\sin x=-\frac{4\,b^2-a^2}{4\,b^2+a^2},\cos x=\frac{4\,a\,b}{4\,b^2+a^2}\right]\right]$$

```
(%i12)  eqns,solns[1],ratsimp;
```
(%o12)
$$[0=0,1=1]$$

两种方法得到的结果实际上是一样的。

十六、　自定义函数

定义一个函数用 `:=` 操作符或者 `define` 函数。一个函数被定义之后，就添加到函数表里。要注意的是这两种定义方法有着微妙的区别，理解它们之间的区别需要用到前面学过的名词和动词的概念。

1　一个误例

```
(%i68)  f(x):=-2*x^3+3*x^2+12*x-13$
(%i69)  fp(x):=diff(f(x),x)$
(%i70)  fp(1);
   ** error while printing error message **
   ~:M: second argument must be a variable; found ~M
   #0: fp(x=1)
    -- an error. To debug this try: debugmode(true);
```

为什么会出错？

出错提示告诉我们，我们实际上在做 `diff(f(1),1)` 这样的运算，也就是说变量代换发生在微分之前。然而我们希望的是，先进行微分运算，再进行变量代替。我们举一个简单的例子

```
(%i1)  f(y):=y^3;
(%o1)
```

$$f\left(y\right):=y^3$$

```
(%i2)  f(z);
(%o2)
```

$$z^3$$

```
(%i3)  f1(x):=diff(f(x),x);
(%o3)
```

$$\mathrm{f1}\left(x\right):=\mathrm{diff}\left(f\left(x\right),x\right)$$

```
(%i4)  f2(x):=''(diff(f(x),x));
(%o4)
```

$$\mathrm{f2}\left(x\right):=3\,x^2$$

```
(%i5)  define(f3(x),diff(f(x),x));
(%o5)
```

$$\mathrm{f3}\left(x\right):=3\,x^2$$

```
(%i6)  [f1(z),f2(z),f3(z)];
(%o6)
```

$$\left[3\,z^2,3\,z^2,3\,z^2\right]$$

```
(%i7)  [f1(1),f2(1),f3(1)];
   ** error while printing error message **
   ~:M: second argument must be a variable; found ~M
   #0: f1(x=1)
    -- an error. To debug this try: debugmode(true);
(%i8)  [f2(1),f3(1)];
(%o8)
```

$$\left[3,3\right]$$

只要 z 没有被赋值，三个微分函数的结果都是正确的。但是当对具体的某个数进行运算时，只有后两个函数返回正确的结果。

```
(%i1)  dplay(g,a,b):=block([val1,val2],
   local(dg),
   define(dg(y),diff(g(y),y)),
   val1:g(a)+dg(a),
   val2:g(b)+dg(b),
   display(val1,val2),
   val1+val2)$
(%i2)  g(x):=x;
(%o2)
```

$$g(x) := x$$

```
(%i3)  dplay(g,1,1);
   val1=2
   val2=2
(%o3)
```

$$4$$

```
(%i4)  g(x):=x^2;
(%o4)
```

$$g(x) := x^2$$

```
(%i5)  dplay(g,1,2);
   val1=3
   val2=8
(%o5)
```

$$11$$

```
(%i6)  dg(3);
(%o6)
```

$$dg(3)$$

上例中的内部声明（local）使 dg 函数只能被 **dplay 本身及被其直接或间接调用的函数**使用。我们再来通过下面的例子强调一下名词性定义:= 与动词性定义 define 的区别

```
(%i1)  f1(x):=print(x)$
(%i2)  define(f2(x),print(x))$
   x
```

从上面我们看出，名词性定义在定义时并没有执行右边的操作，将右边直接作为左边的定义。而动词性定义则即时执行了定义体的操作，将返回的结果作为左边的定义

```
(%i3)  fundef(f1);
(%o3)
```

$$f1(x) := print(x)$$

```
(%i4)  fundef(f2);
(%o4)
```

$$\mathrm{f2}\,(x) := x$$

其中 `fundef` 的作用的是查看非内置函数的定义。

2　无名函数

无名函数并没有什么神奇的，不过没有名字而已，因此必须即时定义即时使用。

```
lambda(arglist,expr1,expr2,...,exprn)
```

上式中，`arglist` 是参数表，而 `expr1,expr2,...,exprn` 是由参数表中的参数构成的函数体，除此之外，还可以用 `%%` 代指前面的结果。函数体顺次执行，并返回最后一个语句 `exprn` 的执行结果

```
(%i1)  [functions,values];
(%o1)
```

$$[[\,],[\,]]$$

```
(%i2)  f:lambda([x,y],x^2+y^3);
(%o2)
```

$$\lambda\left([x,y],x^2+y^3\right)$$

```
(%i3)  [f(2,a),apply(f,[1,2])];
(%o3)
```

$$\left[a^3+4,9\right]$$

```
(%i4)  [functions,values];
(%o4)
```

$$[[\,],[f]]$$

```
(%i5)  map(lambda([x],x^2+sin(x)),[1,2,3]);
(%o5)
```

$$[\sin 1+1,\sin 2+4,\sin 3+9]$$

```
(%i6)  [lambda([x],x+1)(3),lambda([x],x+1,x+2)(3),lambda([x],x+1,%%+2)(3)];
(%o6)
```

$$[4,5,6]$$

无名函数的使用虽然烦琐，但是也有好处——节省了符号，函数随定义随使用。如果函数只是一次使用，为避免名称冲突，就可以使用无名函数。把无名函数赋给一个符号纯属无聊之举。例如 `f:lambda([x,y],x^2+y^3)` 就相当于 `f(x,y):=x^2+y^3`，只不过原来占用函数表现在占用变量表，根本不能体现无名函数的价值。

3　递归函数

下面定义一个计算阶乘的函数

```
(%i1)  myfac(n):=if n=0 then 1 else n*myfac(n-1)$
(%i2)  map('myfac,[0,1,2,3,4]);
(%o2)
```

$$[1,1,2,6,24]$$

```
(%i3)  [0!,1!,2!,3!,4!];
(%o3)
```

$$[1,1,2,6,24]$$

```
(%i4)  map('factorial,[0,1,2,3,4]);
(%o4)
```

$$[1,1,2,6,24]$$

4　勒让德多项式

勒让德多项式 $P_n(x)$[3]的递归定义，可以用 `legendre_p(n,x)` 函数检验，这个函数由正交多项式程序包（`orthogonal polynomial package`）提供。

```
(%i1)  p(n,x):=if n=0 then 1 elseif n=1 then x else
  expand(((2*n-1)/n)*x*p(n-1,x)-((n-1)/n)*p(n-2,x))$
(%i2)  [p(0,x),p(1,x),p(2,x),p(3,x)];
(%o2)
```

$$\left[1, x, \frac{3\,x^2}{2}-\frac{1}{2}, \frac{5\,x^3}{2}-\frac{3\,x}{2}\right]$$

```
(%i3)  map(lambda([nn],p(nn,x)),[0,1,2]);
(%o3)
```

$$\left[1, x, \frac{3\,x^2}{2}-\frac{1}{2}\right]$$

5　典列

典列分索引和值两部分，通过索引可以查到值。典列在其他编程语言里也常出现，在 `Maxima` 里，表现为下标函数：索引是下标，给出下标，就可以根据下标函数计算出函数值。下标函数实际上就是典列的索引与值之间的关联。

静态典列词条的数目是不变的，编号从零开始。而动态典列每一次新的查询，词条总数就增加一

```
(%i1)  (f[k](x):=x^k+1,arrays);
(%o1)
```

$$[f]$$

[3] $P_0=1, P_1=x, P_n(x)=(2n-1)xP_{n-1}(x)-(n-1)P_{n-2}(x)$

```
(%i2)  (makelist(f[n](y),n,0,3),arrayinfo(f));
(%o2)
```

$$[\text{hashed}, 1, [0], [1], [2], [3]]$$

```
(%i3)  (h[n]:=lambda([x],x^n+1),arrays);
(%o3)
```

$$[f, h]$$

```
(%i4)  arrayinfo(h);
(%o4)
```

$$[\text{hashed}, 1]$$

```
(%i5)  (h[2],arrayinfo(h));
(%o5)
```

$$[\text{hashed}, 1, [2]]$$

```
(%i6)  h[2](y);
(%o6)
```

$$y^2+1$$

```
(%i7)  map(lambda([m],h[m](x)),[0,1,2,3]);
(%o7)
```

$$\left[2, x+1, x^2+1, x^3+1\right]$$

```
(%i8)  arrayinfo(h);
(%o8)
```

$$[\text{hashed}, 1, [0], [1], [2], [3]]$$

可以看到，动态典列的生成不需要声明，而静态函数必须要事先声明维数、条目数，这个任务由函数 `array` 完成。函数 `array` 既可以声明一个静态典列，也可以将典列由动态转化为静态。典列最多可以有五维，下面将典列静态化，并定义一个 10×10 的二维典列

```
(%i1)  (array(f,10),arrayinfo(f));
(%o1)
```

$$[\text{declared}, 1, [10]]$$

```
(%i2)  (array(HH,10,10),HH[n,m]:=n^m,HH[2,8]);
(%o2)
```

$$256$$

6　典列的递归

我们尝试修改下标函数

```
(%i1)  (a[n]:=n*a[n-1],a[0]:1,arrayinfo(a));
(%o1)
```

$$[\text{hashed}, 1, [0]]$$

```
(%i2)  [a[4],arrayinfo(a)];
(%o2)
```

$$[24, [\text{hashed}, 1, [0], [1], [2], [3], [4]]]$$

```
(%i3)  (a[n]:=n/2,[a[4],a[7],arrayinfo(a)]);
(%o3)
```

$$\left[24, \frac{7}{2}, [\text{hashed}, 1, [0], [1], [2], [3], [4], [7]]\right]$$

```
(%i4)  (HH[n,m]:=n*m,HH[2,8]);
(%o4)
```

$$16$$

从上面可以看到，对于动态典列，修改下标函数只影响新词条，旧词条保持本来值不改变，我们把这种性质称为动态典列的**保值性**。而对于静态典列，修改下标函数则会立即更新所有词条。

依照上面阶乘函数的定义，假如我们计算了 a[50]，我们也就顺带计算了从 a[1] 到 a[50] 的所有条目，那么再求 a[29] 时就可以通过查表直接获得结果，求 a[51] 也只需要一步计算。

我们用动态典列实现勒让德多项式的递归定义

```
(%i1)  (p[n](x):=expand(((2*n-1)/n)*x*p[n-1](x)-((n-1)/n)*p[n-2](x)),
  p[0](x):=1,p[1](x):=x)$
(%i2)  makelist(p[m](x),m,0,4);
(%o2)
```

$$\left[1, x, \frac{3\,x^2}{2}-\frac{1}{2}, \frac{5\,x^3}{2}-\frac{3\,x}{2}, \frac{35\,x^4}{8}-\frac{15\,x^2}{4}+\frac{3}{8}\right]$$

```
(%i3)  [p[3](x),p[3](y)];
(%o3)
```

$$\left[\frac{5\,x^3}{2}-\frac{3\,x}{2}, \frac{5\,y^3}{2}-\frac{3\,y}{2}\right]$$

```
(%i4)  map(lambda([n],p[n](x)),[0,1,2,3]);
(%o4)
```

$$\left[1, x, \frac{3\,x^2}{2}-\frac{1}{2}, \frac{5\,x^3}{2}-\frac{3\,x}{2}\right]$$

我们用静态典列实现斐波那契数列的定义

```
(%i1)  (array(Fib,10),(Fib[0]:1,Fib[1]:1,Fib[n]:=Fib[n-1]+Fib[n-2]))$
```